基於 SDN 的資料中心大象流負載平衡

Load Balancing for Elephant Flows in SDN-based Data Center

指導教授：蔡孟勳
專題成員：尹子維
開發工具：Python RYU controller, Mininet
測試環境：Linux Ubuntu20.04

一、簡介：

在過去對於基於 SDN 在資料中心的負載平衡的研究中，有針對大象流在資料中心中以分流的方式做負載平衡，也有人提出單純以 Dijkstra's Algorithm 計算出最短路徑，兩者皆是以更新 flow table 的方式來實現負載平衡，大象流的做法需要對所有可能路徑以 link utilization 計算各個路徑的權重在近一步地做分流，而單純以 Dijkstra's Algorithm 做計算路徑會簡易許多也快速許多，因此若將兩者結合，會面臨到的問題是在什麼狀況下應該以大象流分流的算法來達到負載平衡，還是單純以 Dijkstra's Algorithm 來做到負載平衡會更好，換而言之，就是如何知道網路中有較大的流量需要以分流的方式來做到負載平衡。在針對大象流做分流的論文中，作者提出的做法是給予一個閾值來判斷一個 flow 使否為大象流，我希望可以提出一個作法是可以綜觀整體網路的使用狀況，來判斷此時 SDN Controller 應該以分流的方式來解決負載平衡的問題，還是可以更有效率的 Dijkstra's Algorithm 來解決負載平衡的問題。

因此，我的作法是以所有 link utilization 來配適一個 beta 分配，以 beta 分配來的中位數和 link utilization 的平均做比較，如果平均大於中位數，那麼判斷可能整體網路中有很多較大的流量存在，應以分流的做法來實現，反之，若中位數大於平均，那麼可能僅以 Dijkstra's Algorithm 來選擇路徑就好。後者可以更快的反應網路的狀況，前者可以確實地做到讓一個 link 中的大象流被分開。

我的實驗環境是以 Mininet 建立一個 4-ary fattree 的資料中心網路拓撲，在 Ryu controller 中，執行一個 monitor 的程式來擷取網路中的資訊，和一個 ofctl-rest 的程式來接收外部的應用程式計算出的路徑。(如圖1所示)

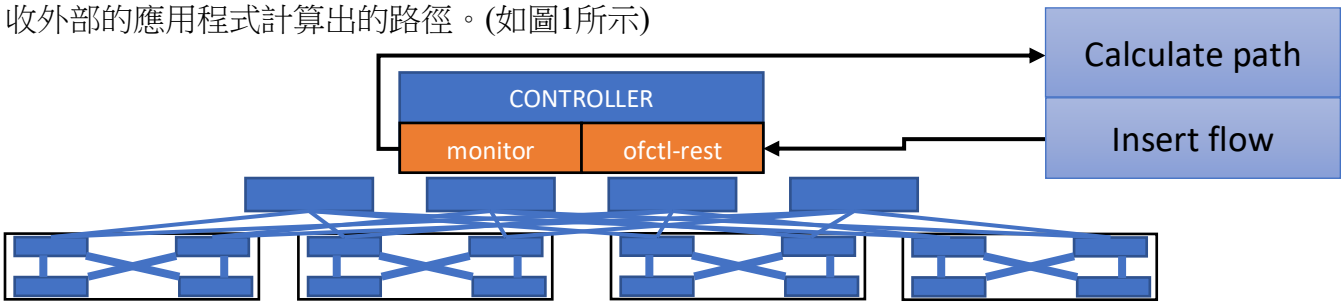

圖1 程式架構

二、測試結果：

測試結果分成 Core switch 連結到 Aggregate switch 的 link 和 Aggregate switch 連結到 Edge switch 的部分，我會挑選連續五個時間點展示結果。

由圖 2 Core 到 Aggregate 的圖展示，在完全沒執行負載平衡的兩個 link 上會出現極高的使用率和極低的使用率同時出現的狀況，而 Dijkstra's 的作法因為是以更換路線的方

式因此在某幾個時間點會出現完全未使用的狀況，相較於分流的作法會有較大的波動，我們的方法相較於分流的作法也會出現波動，會在單純分流的作法的曲線附近震盪，但由於這裡的 bandwidth 比較大，三個方法都能夠解決負載平衡的問題，因此效果並不顯著。

由圖 3 Aggregate 到 Edge 的圖展示，由於這裡的 bandwidth 較小，因此整體的使用率會偏高，可以明顯地看出分流的作法會優於 Dijkstra's 的作法，而我們的方法比起但純使用分流的作法，會有較大的波動，但是從時間點 3 和 4 可以看到在 Dijkstra's 被選擇來做負載平衡時，會進一步的變得平衡一些，在時間點 4 到 5 時，出現較大的流量，分流的方式也會重新介入。

由圖 4 可以看到在 Dijkstra's 的作法上確實更有效率，會使用較少的 CPU 資源。

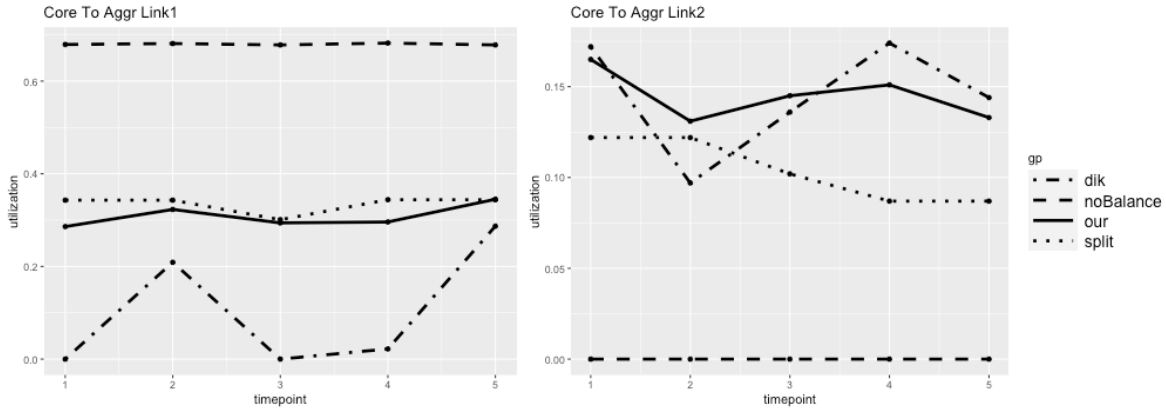

圖 2 Core Switch 到 Aggregate Switch 的 Link utilization

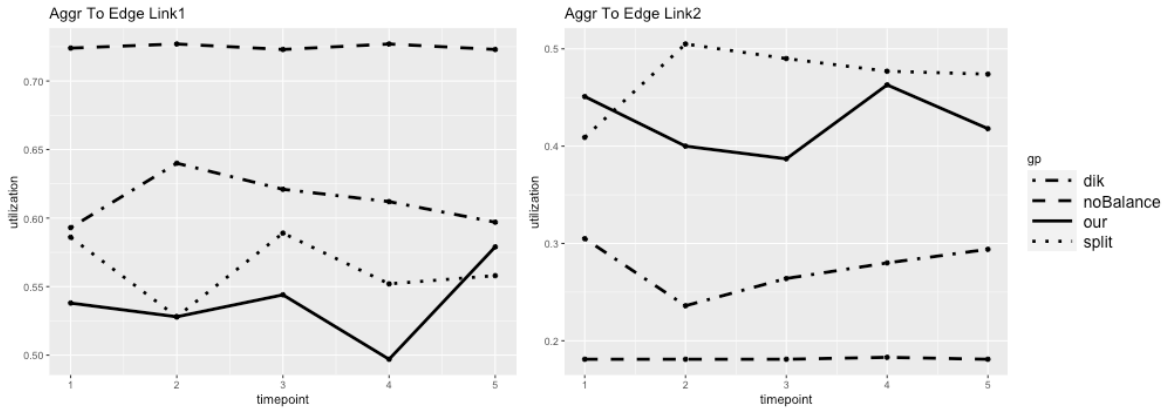

圖 3 Aggregate Switch 到 Edge Switch 的 Link Utilization

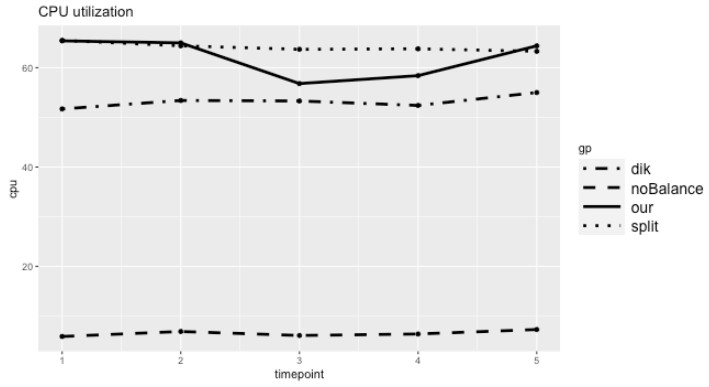

圖 4 四個方法 CPU 的使用率比較